Improved Mixed-Integer Programming Models for Multiprocessor Scheduling with Communication Delays

Sven Mallach

Institut für Informatik
Universität zu Köln, 50923 Köln, Germany

September 29, 2016

Abstract

We revise existing and introduce new mixed-integer programming models for the Multiprocessor Scheduling Problem with Communication Delays. At first, we show how to provably reduce the number of product variables necessary to explicitly linearize the so-called packing formulation that contains bilinear terms. Then, we reveal that the feasible region of almost all existing formulations contains redundant solutions and formulate new constraints in order to exclude these. As a result and by exploiting further structural properties, the models are improved in their strength and, at the same time, in terms of their size and modeling complexity. This inevitably leads to new more compact formulations which are then experimentally compared with each other and with further formulations from the literature. We set up a realistic scenario with a preprocessing of the task graphs, delivering the gained information equally to all the tested models and evaluate not only running times but also the obtained lower and upper bounds on the makespan objective for unsolved instances of a large scale benchmark set.

1 Introduction

The Multiprocessor Scheduling Problem with Communication Delays (MPSCD) is to find a minimum makespan schedule of a task system with precedence constraints on a set of homogeneous processors subject to delays that arise if dependent tasks are processed on different processors. More formally, let $G = (V, A)$ be a task graph where $(i, j) \in A$ if task j needs to be processed after task i has finished. Each task $i \in V$ is associated with a fixed processing time L_i that is independent of the processor assigned to i. However, if $(i, j) \in A$ and the two tasks are processed on different processors, then j can start no earlier than at time $t_i + L_i + c_{ij}$ where t_i denotes the starting time of task $i \in V$ and c_{ij} the communication delay between i and j.

The MPSCD has obvious applications in scheduling tasks on fully connected network multiprocessor environments. Using the widely accepted notation proposed in [15], it can be described as $P|\text{prec}, c_{ij}|C_{max}$. It is a generalization of the

classical multiprocessor scheduling problem without precedence constraints and communication delays $P|\epsilon|C_{max}$ which is already $\mathcal{NP}$-hard even if the number of processors P is equal to two [13].

Related Work. There is a lot of research that deals with the hardness of the problem and its variants. Rayward-Smith showed that the preemptive version of the problem is $\mathcal{NP}$-hard for any fixed communication delays larger than one [29] and that the non-preemptive case is $\mathcal{NP}$-hard even if both, the processing times and the communication delays, are of unit length [30]. Further complexity results and algorithms for several special cases of the problem are discussed in [36, 37], and surveyed in [4] and in [14].

For the general problem setting, there is an approximation algorithm called ETF (Earliest Task First) [16] whose approximation factor can be bounded by the communication costs along some predecessor-successor-chains of tasks in the computed schedule. A $\frac{7}{3}$-approximation algorithm can be constructed when all the communication delays c_{ij} are restricted to be no larger than the processing times of immediate predecessors of j and immediate successors of i [28]. A common heuristic idea is to extend usual multiprocessor list or level scheduling algorithms to deal with communication delays. For example, the MCP heuristic by Wu and Gajski [40] and the DLS algorithm by Sih and Lee [35] fall into this category. Yang and Gerasoulis discuss the differences and performance issues when communication delays are considered in list scheduling heuristics and provide further variants in [42]. Later, Djordjević and Tošić [9] present a single-pass heuristic algorithm which is a generalized list scheduling technique called *chaining*. Kwok et al. combine list scheduling with a local search step in [24] and the ISH algorithm [20] inserts tasks into idle intervals that can occur in ordinary list scheduling. The mapping heuristic [10], the DCP heuristic [22], the LAST heuristic by Baxter and Patel [3], the bottom-up technique by Mehdiratta and Ghose [27], the two-pass technique by [18], and the bubble algorithm by Kwok and Ahmad [21] are further proposed algorithms. There are also some heuristic algorithms that use task duplication, i.e., tasks may be executed by more than one processor in order to reduce communication delays. One instance of these is the DSH algorithm by [20]. Another heuristic strategy is to decompose the problem into two stages, namely the clustering of tasks to be processed on the same processor and the subsequent ordering of these clusters. This approach has been used, e.g., by Kim and Browne [19], Sarkar [31], Yang and Gerasoulis [41, 43], McCreary and Gill [26], and Senar et al. [33].

Fujita and Yamashita survey the connections between the list scheduling algorithm variants with and without communication delays, the clustering-based algorithms and some of the complexity and approximability results in [12]. Another even more comprehensive survey presenting the many different approaches until 1999 and establishing a partial classification taxonomy is given by Kwok and Ahmad in [23]. An experimental comparison of several heuristic approaches has been carried out by Jin et al. [17]. Kong et al. applied partical swarm optimization to the problem.

On the exact side, there is an A^*-based algorithm to solve the problem to optimality [34] that can however suffer from large memory requirements for instances with 40 or more tasks [39]. Satish et al. [32] give a Benders-decomposition based constraint programming algorithm that quickly finds feasible schedules also for larger instances with up to 100 tasks. However, optimality can be proven only sometimes within a short time frame and rather for the

smaller of the tested instances. The majority of approaches to solve the problem exactly is based on mixed-integer programming (MILP) formulations. The first effort in this direction was given by Davidović et al. in [8] and [6], and has later been improved by the authors in [7]. In subsequent research, the latter model is usually called the *packing formulation*. It contains bilinear terms in order to incorporate the communication delays which can be linearized either in a standard or in a more sophisticated manner [7, 25]. The packing formulation has been subject to several revisions by Venugopalan and Sinnen in [38] and [39]. The modifications focus mainly on avoiding the binary products beforehand by different modeling tricks, and on reducing the total number of constraints. Another proposal into this direction has been given by Ait El Cadi et al. [2]. Until today, the different integer programming models could also only be used to solve smaller instances with up to 50 tasks.

Contribution. In this paper, we present improvements applicable to almost all of the integer programming models known so far. These yield advantages in their size, strength, and modeling complexity. First, we show how to provably reduce the number of product variables necessary to explicitly linearize the packing formulation. Then, we prove a structural property that can be used to eliminate symmetric and redundant solutions from the search space of all the models as presented in [7, 38, 39]. As a side product, we briefly review the many different models and their variants from the literature using a unified notation. We identify two major modeling strategies and let our and previous improvements meld into two new more compact and strengthened formulations. These are then experimentally compared with the improved linearized packing formulation and the model by Ait El Cadi et al. from [2] using a large set of benchmark instances with up to 100 tasks. The experiments are carried out in a realistic setting where all of the models are initialized with the same preprocessing information as it would naturally be computed when solving the problem in practice. Our results show that, when a standard commercial MILP solver is used, the performance of the models varies significantly in terms of (a) finding good feasible solutions quickly and (b) proving optimality of known solutions by deriving strong lower bounds on the makespan. In these disciplines and hence also in the number of solved instances (given a time limit of ten minutes), the models derived in this paper compare favorably with the others. The results also indicate that the performance depends on the precedence structure of the task graphs and the number of processors P.

Outline. This manuscript is built up as follows. Sect. 2 gives basic definitions and notations necessary to understand the problem and briefly reviews the most important integer programming models known so far. The proposed improvements are then presented in Sect. 3 and new models that make use of these features are introduced in Sect. 4. The evaluation of these models takes place in Sect. 5 and we close with conclusions in Sect. 6.

2 Problem Statement and Known Formulations

2.1 The Multiprocessor Scheduling Problem with Communication Delays (MPSCD)

The input to the MPSCD consists of a task graph $G = (V, A)$ together with (a) processing times L_i associated to each task $i \in V$, (b) edge weights c_{ij} for each $(i, j) \in A$ specifying the communication delays, and (c) the number P of available processors. If t_i denotes the starting time of a task i and $(i, j) \in A$, then task j can start its execution earliest at time $t_i + L_i$ if j is processed on the same processor as task i. If i and j are processed on different processors, then j can start earliest at time $t_i + L_i + c_{ij}$. The standard and here considered objective of the MPSCD is to minimize the makespan $\max_{i \in V} t_i + L_i$.

As a convention, we will assume that the processors are enumerated by an index set $\mathcal{P} = \{1, \ldots, P\}$. Similarly, the task set is associated with the vertices $V = \{1, \ldots, n\}$ of the task graph.

2.2 The packing formulation

The packing formulation as presented in [7] uses several types of variables, starting with general integer variables (without explicit integrality requirement), t_i and p_i, specifying the starting time and processor index of each task $i \in V$, respectively. The fundamental idea of the formulation is, however, to model a feasible overlapping of tasks using special kinds of ordering variables for all $i, j \in V$, $i \neq j$:

$$\sigma_{ij} = \begin{cases} 1, & \text{if task } i \text{ must finish before task } j \text{ starts} \\ 0, & \text{otherwise} \end{cases}$$

$$\epsilon_{ij} = \begin{cases} 1, & \text{if } p_i < p_j \\ 0, & \text{otherwise} \end{cases}$$

Additionally, the precise processor assignment for each task is reflected using the following variables for each task $i \in V$, and each $k \in \mathcal{P}$:

$$x_{ik} = \begin{cases} 1, & \text{if } p_i = k \\ 0, & \text{otherwise} \end{cases}$$

The packing formulation is the following mixed-integer quadratic program:

$$
\begin{aligned}
&\min C_{max} \\
&s.t. \quad \sigma_{ij} + \sigma_{ji} && \leq 1 && \text{for all } i,j \in V,\ i \neq j && (1)\\
&\quad \epsilon_{ij} + \epsilon_{ji} && \leq 1 && \text{for all } i,j \in V,\ i \neq j && (2)\\
&\quad \sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} && \geq 1 && \text{for all } i,j \in V,\ i \neq j && (3)\\
&\quad \sigma_{ij} && = 1 && \text{for all } (i,j) \in A && (4)\\
&\quad t_i + L_i && \leq C_{max} && \text{for all } i \in V && (5)\\
&\quad t_j - t_i - L_i - (\sigma_{ij} - 1)M && \geq 0 && \text{for all } i,j \in V,\ i \neq j && (6)\\
&\quad p_j - p_i - 1 - (\epsilon_{ij} - 1)P && \geq 0 && \text{for all } i,j \in V,\ i \neq j && (7)\\
&\quad t_i + L_i + \sum_{k \in \mathcal{P}} \sum_{l \in \mathcal{P}, k \neq l} c_{ij} x_{ik} x_{jl} && \leq t_j && \text{for all } (i,j) \in A && (8)\\
&\quad \sum_{k \in \mathcal{P}} x_{ik} && = 1 && \text{for all } i \in V && (9)\\
&\quad \sum_{k \in \mathcal{P}} k\ x_{ik} && = p_i && \text{for all } i \in V && (10)\\
&\quad C_{max} && \geq 0 \\
&\quad t_i && \geq 0 && \text{for all } i \in V \\
&\quad p_i && \geq 1 && \text{for all } i \in V \\
&\quad x_{ik} && \in \{0,1\} && \text{for all } i \in V,\ k \in \mathcal{P} \\
&\quad \sigma_{ij} && \in \{0,1\} && \text{for all } i,j \in V,\ i \neq j \\
&\quad \epsilon_{ij} && \in \{0,1\} && \text{for all } i,j \in V,\ i \neq j
\end{aligned}
$$

Inequalities (1) and (2) impose that there can be at most one valid ordering of tasks and their assigned processor indices, respectively. Further, if an ordering is already known due to the precedence relations, then it is set by (4).

If two tasks i and j are running on the same processor ($\epsilon_{ij} + \epsilon_{ji} = 0$), then inequalities (3) make sure that they are ordered. Similarly, if neither task starts before the other finishes ($\sigma_{ij} + \sigma_{ji} = 0$), then they must be running on different processors.

The constraints (5) impose the correct lower bounds on the makespan objective variable to be minimized.

Inequalities (6) make sure that the starting times of ordered tasks are consistent. The constant M needs to be chosen such that the constraint is satisfied if j is not ordered after i ($\sigma_{ij} = 0$). In [7], it is proposed to set M to any known upper bound on the makespan which is safe but unnecessarily weakens the formulation as will be discussed in Sect. 3.3.

If ϵ_{ij} is set to 1, inequalities (7) require p_j to be strictly greater than p_i. However, in principle, the formulation allows ϵ_{ij} and ϵ_{ji} to be both zero while $p_i \neq p_j$. This causes no harm since the communication delays are imposed based on the x-variables. However, it is of interest for different modeling approaches as discussed in Sect. 2.3.2.

Inequalities (8) impose the precedence constraints and communication delays if the respective tasks are assigned to different processors. In the displayed form, these constraints have bilinear terms making the formulation a non-linear one. We will discuss in the following subsection how to deal with this issue.

Finally, each task must be assigned to exactly one processor as is enforced by (9) and (10) translates these decisions into the correct processor indices.

2.3 Linearization of the packing formulation

What has been left open so far is how to deal with the products arising in constraints (8). Different approaches have been proposed in the literature and will be discussed in the following. We will distinguish explicit linearizations where the bilinear terms are replaced by additional variables, and implicit linearizations where the products are avoided by modeling the problem differently.

2.3.1 Explicit linearization of the packing formulation

The most straightforward way to linearize the bilinear terms is to apply the standard linearization approach used in integer linear programming as it has been originally suggested by Fortet [11] and has been carried out for this problem in [7]. Each product $x_{ik}x_{jl}$ is modeled using a variable y_{ij}^{kl} and three constraints:

$$y_{ij}^{kl} \leq x_{ik} \qquad \text{for all } (i,j) \in A \; k,l \in \mathcal{P}, k \neq l \tag{11}$$

$$y_{ij}^{kl} \leq x_{jl} \qquad \text{for all } (i,j) \in A \; k,l \in \mathcal{P}, k \neq l \tag{12}$$

$$y_{ij}^{kl} \geq x_{ik} + x_{jl} - 1 \qquad \text{for all } (i,j) \in A \; k,l \in \mathcal{P}, k \neq l \tag{13}$$

The second one is an application of a more involved linearization method (called *compact linearization*). It can be applied only to integer programs with assignment constraints and has been proposed by Liberti [25]. Following [7], one needs to add the constraints:

$$\sum_{k \in \mathcal{P}} y_{ij}^{kl} = x_{jl} \qquad \text{for all } i,j \in V, i \neq j, \; l \in \mathcal{P} \tag{14}$$

$$y_{ij}^{kl} = y_{ji}^{lk} \qquad \text{for all } i,j \in V, i \neq j, \; k,l \in \mathcal{P} \tag{15}$$

Clearly, (15) needs to be stated only for the cases where $i < j$. According to the experimental remarks given in [7], each of the two strategies may be superior in practice and this depends mainly on the density of the task graph. An important fact to notice here is that the 'compact' linearization approach requires y-variables to be introduced for each pair of tasks, not only for dependent ones.

2.3.2 Implicit linearization of the packing formulation

Venugopalan and Sinnen [38, 39] proposed three linear variants of the packing formulation that avoid the introduction of additional variables.

The idea of the first approach is to plug in constraints (13) directly into inequalities (8), leading to (16). In order to preserve validity in the case that $x_{ik} + x_{jl} = 0$, inequalities (17) are added.

$$t_i + L_i + c_{ij}(x_{ik} + x_{jl} - 1) \leq t_j \qquad \text{for all } (i,j) \in A, \; k,l \in \mathcal{P}, k \neq l \tag{16}$$

$$t_i + L_i \leq t_j \qquad \text{for all } (i,j) \in A \tag{17}$$

Alternatively, the authors propose to model the communication delays using the ϵ-variables by replacing inequalities (8) with (18). In [38], they enforce transitive consistency on the ϵ-variables by inequalities (19). This is however not sufficient to also achieve consistency with the p-variables as this still allows

for cases where $p_i \neq p_j$ although $\epsilon_{ij} = \epsilon_{ji} = 0$. Further, the number of these constraints is of order $\Theta(|V|^3)$.

$$t_i + L_i + c_{ij}(\epsilon_{ij} + \epsilon_{ji}) \leq t_j \qquad \text{for all } (i,j) \in A \tag{18}$$

$$\epsilon_{ij} + \epsilon_{jk} \geq \epsilon_{ik} \qquad \text{for all } i,j,k \in V,\ i \neq j \neq k \tag{19}$$

Both problems can however be circumvented. Venugopalan and Sinnen showed in [39] that consistency of the processor assignments, without the x-variables, can also be established using inequalities (20) instead of (19).

$$p_j - p_i - \epsilon_{ij} P \leq 0 \qquad \text{for all } i,j \in V,\ i \neq j \tag{20}$$

The total number of constraints is then only of order $\mathcal{O}(|V|^2)$. Moreover, the x-variables together with constraints (9) and (10) can be dropped when using this strategy which lets the size of the formulation be independent from the number of processors.

Another approach that requires no more than a quadratic number of variables and constraints in the number of tasks and processors is given by Ait El Cadi et al. [2]. It uses only the x-, σ- and t-variables and the constraints (9) and (5) from the packing formulation. Further, the constraints (8) are reformulated by removing the summations and adding a constraint for each pair of processors instead, using individual coefficients $c_{ij,kl}$ for each pair of processors $k,l \in \mathcal{P}$ (so that $c_{ij,kl}$ can be set to 0 for $k = l$):

$$t_i + L_i + c_{ij,kl}(x_{ik} + x_{jl} - 1) \leq t_j \qquad \text{for all } (i,j) \in A,\ k,l \in \mathcal{P}$$

Independent tasks are ordered using the following big-M constraints:

$$t_i + L_i + M(3 - x_{ik} + x_{jk} - \sigma_{ij}) \leq t_j \quad \text{for all } i,j \in V,\ (i,j),(j,i) \notin A, k \in \mathcal{P}$$

$$t_j + L_j + M(2 - x_{ik} + x_{jk} + \sigma_{ij}) \leq t_i \quad \text{for all } i,j \in V,\ (i,j),(j,i) \notin A, k \in \mathcal{P}$$

3 New Reductions and Improvements

3.1 Compacting the compact linearization

As stated in Sect. 2.3.1, when applying the compact linearization for the packing formulation as proposed in [7], the number of variables increases over the standard linearization method. While in the usual approach, $|A| \cdot P(P-1)$ variables and $3|A| \cdot P(P-1)$ inequalities need to be introduced, the compact method requires $|V|(|V|-1) \cdot P^2$ variables, $|V|(|V|-1) \cdot P$ equations of type (14) and (with the obvious reduction) $|V|(|V|-1)/2 \cdot P^2$ equations of type (15).

From equations (15) one can see directly that half of the variables are in fact redundant. We can eliminate all z-variables for $j > i$ and all equations (15) by reformulating and splitting (14) as follows:

$$\sum_{k \in \mathcal{P}} y_{ij}^{kl} = x_{jl} \qquad \text{for all } i,j \in V, i < j,\ l \in \mathcal{P} \tag{21}$$

$$\sum_{h \in \mathcal{P}} y_{ji}^{lk} = x_{jl} \qquad \text{for all } i,j \in V, i > j,\ l \in \mathcal{P} \tag{22}$$

However, since the products are only needed (and evaluated) for dependent task pairs, we strive to make the number of additional variables and equations proportional to the number of arcs again. We proceed as follows.

Given $(i,j) \in A$, we are interested in all the products $x_{i,k} \cdot x_{j,l}$ for all $k, l \in \{1, \ldots, P\}$. For each vertex $i \in V$, let $A_i = \{j \in V \mid (i,j) \in A \text{ or } (j,i) \in A\}$. As a consequence, for $(i,j) \in A$, it holds that $i \in A_j$ and $j \in A_i$.

Consider the set of equations:

$$\sum_{k \in \mathcal{P}} x_{ik} x_{jl} = x_{jl} \qquad \text{for all } i \in V,\ j \in A_i,\ l \in \mathcal{P} \tag{23}$$

We want to introduce y_{ij}^{kl} only for $i < j$ (irrelevant whether $(i,j) \in A$ or $(j,i) \in A$). Hence, we replace each equation (23) by one of the following two equations, depending on whether $i < j$ or $j < i$:

$$\sum_{k \in \mathcal{P}} y_{ij}^{kl} = x_{jl} \qquad \text{for all } i \in V,\ j \in A_i,\ i < j,\ l \in \mathcal{P} \tag{24}$$

$$\sum_{k \in \mathcal{P}} y_{ji}^{lk} = x_{jl} \qquad \text{for all } i \in V,\ j \in A_i,\ j < i,\ l \in \mathcal{P} \tag{25}$$

Theorem 3.1. *For any integral solution to the x-variables satisfying constraints (9), equations (24) and (25) imply that $y_{ij}^{kl} = x_{ik} \cdot x_{jl}$ for each $(i,j) \in A$ and each $k, l \in \{1, \ldots, P\}$.*

Proof. We pick some $(i,j) \in A$ and assume, w.l.o.g., that $i < j$ such that we have a set of equations of type (24) (however, for $j < i$, the following arguments can equally be stated using equations (25)). Let the rows of this equation set be ordered in the following way:

$$\begin{array}{llll} y_{ij}^{11} & + y_{ij}^{21} & + \ldots & + y_{ij}^{P1} = x_{j1} \\ y_{ij}^{12} & + y_{ij}^{22} & + \ldots & + y_{ij}^{P2} = x_{j2} \\ \ldots & + \ldots & + \ldots & + \ldots = \ldots \\ y_{ij}^{1P} & + y_{ij}^{2P} & + \ldots & + y_{ij}^{PP} = x_{jP} \end{array}$$

Now, since $x_{j1} + x_{j2} + \cdots + x_{jP} = 1$, exactly one of the above rows has a right hand side of 1. Let the l-th row be the according one, i.e.,

$$y_{ij}^{1l} + y_{ij}^{2l} + \ldots + y_{ij}^{Pl} = x_{jl} = 1$$

For a moment, we assume integrality of the y-variables and hence, one of the variables on the left hand side must be equal to 1. This is semantically correct: Exactly one product variable in the corresponding row must be 1, since also $x_{i1} + x_{i2} + \cdots + x_{iP} = 1$. Now suppose that $y_{ij}^{kl} = 1$ for some $k \in \mathcal{P}$. If then $x_{ik} = 1$, this would be fine, so assume to the contrary that $x_{ik} = 0$. Consider then the following row that exists since $i \in A_j$:

$$y_{ij}^{k1} + y_{ij}^{k2} + \ldots + y_{ij}^{kP} = x_{ik} = 0$$

The left hand side of the latter equation contains y_{ij}^{kl} which is assumed to be 1, so x_{ik} cannot be 0 at the same time - a contradiction. So until here, we can conclude that whenever $y_{ij}^{kl} = 1$ then $x_{ik} = x_{jl} = 1$ as desired.

Assume now that $y_{ij}^{kl} = 0$. If this is because the row containing y_{ij}^{kl} has right hand side $x_{jl} = 0$, then this is clearly correct. So assume that $x_{jl} = 1$. In this case, there must be some $y_{ij}^{k'l} = 1$ for some $k' \neq k$, and, hence by the observations just made, this implies $x_{ik'} = 1$. As a consequence, by constraints (9), $x_{ik} = 0$ as desired.

By a similar argument, one can show that integrality of the x-variables implies integrality of the y-variables. Suppose that the right hand side 1 in

$$y_{ij}^{1l} \quad + y_{ij}^{2l} \quad + \ldots \quad + y_{ij}^{Pl} \quad = x_{jl} = 1$$

is split over multiple variables, say w.l.o.g. over y_{ij}^{al} and y_{ij}^{bl}. Then there are rows

$$\begin{array}{llll} y_{ij}^{a1} & + y_{ij}^{a2} & + \ldots & + y_{ij}^{aP} \quad = x_{ia} \\ y_{ij}^{b1} & + y_{ij}^{b2} & + \ldots & + y_{ij}^{bP} \quad = x_{ib} \end{array}$$

where the upper one contains $y_{ij}^{al} > 0$ and the lower one contains $y_{ij}^{bl} > 0$ on the left hand sides. This means that both x_{ia} and x_{ib} are forced to be > 0 which is, again by (9), impossible. □

Corollary 3.2. *There is a compact linearization of the packing formulation that requires only $|A| \cdot P^2$ additional variables and $2|A|P$ additional constraints.*

3.2 Excluding Redundant Integer Solutions

Let $T_P = \{t \in \mathbb{N}^V \mid t \text{ feasible}\}$ be the set of feasible task starting time vectors where there exists a feasible processor assignment for P processors.

The set of inequalities (1), (2) and (3) allows for multiple solutions in the σ- and ϵ-variables that correspond to the same task starting time vectors $t \in T_P$. Before we prove this, we state some central observations that help us afterwards.

Observation 3.3. *Since $\sigma_{ij} = 1$ for all $(i,j) \in A$, inequalities (3) are trivially satisfied for dependent pairs of tasks and can hence be omitted for them. The same is true for (1) since $\sigma_{ji} = 0$ and for (6) since dependent tasks are already ordered due to the communication delay constraints.*

Observation 3.4. *(as already stated in [39]) Any integer solution to inequalities (6) implies (1) to be satisfied. Further, any integer solution to (7) and (20) implies (2) to be satisfied.*

Theorem 3.5. *Let $i, j \in V$ such that neither $(i,j) \in A$ nor $(j,i) \in A$. Then, for each $t \in T_P$, there is a feasible solution that satisfies:*

$$\sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} = 1 \tag{26}$$

Proof. We need to show that an exclusion of the case where $\sigma_{ij} + \sigma_{ji} = 1$ and $\epsilon_{ij} + \epsilon_{ji} = 1$ at the same time is permitted, i.e., that adding (26) does not exclude valid solutions in terms of the starting times of the tasks.

An important prerequisite is to see that all independent task-pairs $k, l \in V$ that are assigned to the same processor ($\epsilon_{kl} = \epsilon_{lk} = 0$) are ordered by combining (6) with either (3) or (26). In particular, (6) are the only other constraints where the σ-variables are incorporated. The two instances of (6) for i and j read:

$$t_i \geq t_j + L_j - M(1 - \sigma_{ji}) \tag{6a}$$
$$t_j \geq t_i + L_i - M(1 - \sigma_{ij}) \tag{6b}$$

Since i and j are independent, t_i and t_j only depend on the other tasks assigned to the respective processors of i and j which must be different as we assume that $\epsilon_{ij} + \epsilon_{ji} = 1$. Let, w.l.o.g., $t_i \leq t_j$ and consider the set of predecessors of j on its processor $J_< = \{k \in V \mid \epsilon_{jk} + \epsilon_{kj} = 0,\ \sigma_{kj} = 1\}$. Due

to the ordering of these tasks, there is some minimum possible starting time $t_j^{min} = \max_{k \in J_<} t_k + L_k$ for j. Clearly, $t_j \geq t_j^{min}$.

Since $t_i \leq t_j$ and when reasonably assuming $L_j > 0$, (6a) implies $\sigma_{ji} = 0$. Moreover, any potential feasible solution where j starts in the (possibly empty) interval $[t_j^{min}, t_i + L_i - 1]$ requires $\sigma_{ij} = 0$ in order to not violate (6b) and hence satisfies (26). Finally, for all feasible solutions where $t_j \geq t_i + L_i$, the inequality (6b) is still satisfied (for any $M \geq 0$) if we set σ_{ij} to 0. Hence, adding (26) does not exclude any solutions.

For the symmetric case $t_j \leq t_i$, the same line of argumentation can be used with the roles of i and j, σ_{ij} and σ_{ji}, and (6a) and (6b) interchanged. □

Corollary 3.6. *Using equation* (26) *instead of (1), (2) and (3) yields a stronger formulation and continuous relaxation of the problem.*

Proof. This is immediate, as (26) excludes feasible solutions in terms of the σ-variables that are redundant w.r.t. the feasible starting time vectors t and satisfy (3). Moreover, constraints (1) and (2) are implied by (26) not only for integer but also for fractional solutions. □

3.3 Pair-dependent choice of M

In inequalities (6), the constant M needs to be chosen such that the constraint is satisfied if j is not ordered after i ($\sigma_{ij} = 0$). In [7], it is proposed to set M to any known upper bound on the makespan, and in [39], it is proposed to set it to $\sum_{i \in V} L_i + \sum_{(i,j) \in A} c_{ij}$. Both choices are safe but unnecessarily weaken the formulation w.r.t. fractional solutions in the σ-variables. Especially, since this ordering constraint is only required for independent pairs of tasks, the second summation is superfluous. Moreover, in practice, one will usually compute a heuristic solution in advance, and therefore have not only lower bound positions $\mathrm{lb}(i)$ but also upper bound positions $\mathrm{ub}(i)$ for each task $i \in V$. In this case, it is possible to compute a stronger individual value $M_{ij} = \mathrm{ub}(i) + L_i - \mathrm{lb}(j)$ for each of the constraints (6).

3.4 Redundancy in Task-Processor Assignments

In the original packing formulation, variables p_i were just an alias for $\sum_{k \in \mathcal{P}} k\, x_{ik}$ and all of their occurrences could therefore be replaced by this term. In essence, as already indicated in Sect. 2.3.2, there are two choices that avoid redundancy in this respect. Either one uses the p-variables together with the ϵ-variables and inequalities (20). Or one uses just the x-variables. We will see in Sect. 4.2 how constraints (26) can be reformulated with x-variables. In [39], the authors state that it is an advantage of their model to be completely independent in size from the number of processors. However, if this requires more variable-types whose numbers are quadratic in $|V|$ (like the ϵ-variables), this will typically result in a net increase over $|V| \cdot P$ since one can assume that multiprocessing systems are usually applied to task systems where the number of tasks is as least as large as the number of processors.

4 New Models

A first improved model results when applying all the improvements from Sect. 3 to the packing formulation. In particular, inequalities (1), (2) and (3) are replaced by equation (26), variables σ_{ij} and inequalities (6) (with the task-dependent M_{ij} from Sect. 3.3) are restricted to $i, j \in V$ where neither (i, j) nor $(j, i) \in A$. Finally, the linearization approach from Sect. 3.1 is applied and the bilinear terms $x_{ik}x_{jl}$ are replaced by the corresponding variables y_{ij}^{kl}.

To address the either-or choice between the p and the x-variables as explained in Sect. 3.4, we now formulate two new models that incorporate the reductions from sections 3.2 and 3.3 as well. To stress their reduced complexity and the choice of either p- or x-variables, we call these two models *Core-P* and *Core-X*.

4.1 Formulation Core-P

This formulation is a straightforward application of the changes and reductions from sections 3.2 and 3.3 to the formulation by Venugopalan and Sinnen from [39].

$$
\begin{array}{lllll}
\min & C_{max} & & & \\
s.t. & \sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} & = 1 & \text{for all } i, j \in V,\ i < j,\ (i, j), (j, i) \notin A \\
& t_i + L_i & \leq C_{max} & \text{for all } i \in V \\
& t_j - t_i - L_i - (\sigma_{ij} - 1)M_{ij} & \geq 0 & \text{for all } i, j \in V,\ (i, j), (j, i) \notin A \\
& p_j - p_i - 1 - (\epsilon_{ij} - 1)P & \geq 0 & \text{for all } i, j \in V,\ i \neq j \\
& p_j - p_i - \epsilon_{ij}P & \leq 0 & \text{for all } i, j \in V,\ i \neq j \\
& t_i + L_i + c_{ij}(\epsilon_{ij} + \epsilon_{ji}) & \leq t_j & \text{for all } (i, j) \in A \\
& C_{max} & \geq 0 & \\
& t_i & \geq 0 & \text{for all } i \in V \\
& p_i & \geq 1 & \text{for all } i \in V \\
& \sigma_{ij} & \in \{0, 1\} & \text{for all } i, j \in V,\ (i, j), (j, i) \notin A \\
& \epsilon_{ij} & \in \{0, 1\} & \text{for all } i, j \in V,\ i \neq j
\end{array}
$$

4.2 Formulation Core-X

Formulation *Core-X* is a variant with only x-variables. However, they are used in a different way as proposed in [2] or [38]. Instead of inequalities (16) and (17), we install (28) and (29) to model the communication delays. Inequalities (28) enforce these, whenever i is assigned to processor k and j is assigned to a processor with strictly larger index. Similarly, (29) has the same effect when j is assigned to a processor with strictly smaller index. Further, since there are no ϵ-variables, the new equations (26) from Sect. 3.2 are turned into inequalities (27). These enforce $\sigma_{ij} + \sigma_{ji} = 1$ if and only if i and j are both assigned to processor $k \in \mathcal{P}$, i.e., when $x_{ik} = 1$ and all x_{jl} for $l \neq k$ are zero. For ease of notation, let $V_i = \{j \in V \mid i < j \text{ and } (i, j) \notin A \text{ and } (j, i) \notin A\}$.

$$
\begin{aligned}
&\min C_{max} \\
&s.t.\ \ \sigma_{ij} + \sigma_{ji} - x_{i,k} + \sum_{l \in \mathcal{P}, l \neq k} x_{jl} && \geq 0 && \text{for all } j \in V_i,\ k \in \mathcal{P} && (27)\\
&\quad t_i + L_i && \leq C_{max} && \text{for all } i \in V \\
&\quad t_j - t_i - L_i - (\sigma_{ij} - 1)M_{ij} && \geq 0 && \text{for all } i,j \in V,\ (i,j),(j,i) \notin A \\
&\quad t_i + L_i + c_{ij}(x_{ik} - \sum_{l \in \mathcal{P}, l \leq k} x_{jl}) && \leq t_j && \text{for all } (i,j) \in A,\ k \in \mathcal{P} && (28)\\
&\quad t_i + L_i + c_{ij}(x_{ik} - \sum_{l \in \mathcal{P}, l \geq k} x_{jl}) && \leq t_j && \text{for all } (i,j) \in A,\ k \in \mathcal{P} && (29)\\
&\quad \sum_{k \in \mathcal{P}} x_{ik} && = 1 && \text{for all } i \in V \\
&\quad C_{max} && \geq 0 \\
&\quad t_i && \geq 0 && \text{for all } i \in V \\
&\quad x_{ik} && \in \{0,1\} && \text{for all } i \in V,\ k \in \mathcal{P} \\
&\quad \sigma_{ij} && \in \{0,1\} && \text{for all } i,j \in V,\ (i,j),(j,i) \notin A
\end{aligned}
$$

5 Experimental Evaluation

5.1 MIP models

The developments presented suggest a comparison of the following problem formulations:

- *PCK-C*: The linearized packing formulation with all improvements as mentioned in the beginning of Sect. 4.
- *Core-P*: The new formulation with p- and ϵ-variables (but no x-variables) from Sect. 4.1.
- *Core-X*: The new formulation with x- but neither p- nor ϵ-variables from Sect. 4.2.
- *Core-X-VS*: Like *Core-X*, but with the inequalities (16) and (17) from [38] instead of (28) and (29) to formulate the communication delay constraints.
- *2M*: The formulation by Ait El Cadi et al. from [2].

5.2 Instances

We use three benchmark sets from the literature.

The first set Structure consists of 207 task graphs that have been used for comparisons already in [39]. Here, the emphasis is laid on determining a model's sensitivity to different input graph structures. The instance sets consists of in-trees (abbreviated in figures as InT), out-trees (OutT), fork- (F), join- (J), and fork-and-join (F-J) graphs, graphs without arcs, i.e., only independent tasks (Ind), pipeline graphs (Pipe, with diamond structures), stencil (St), series-parallel (SP) and random (Rnd) graphs. All instances have either 10, 21 or 30

nodes and varying density. Another parameter of interest is the communication-to-computation ratio (CCR) defined as CCR $= \sum_{(i,j)\in A} c_{ij} / \sum_{i\in V} L_i$. The graphs were generated such that the CCR is either about 0.1, 1, 2 or 10.

The second set is from [5] and can be further divided into two subsets. In the first subset (called DC from now on), the instances have a randomly generated precedence structure, random communication delays, and either 50, 100, 200, 300, 400 or 500 nodes. For each of these sizes, there are 30 instances, more precisely each time six with density 20, 40, 50, 60, 80 where density is defined as the percentage of arcs w.r.t. a complete directed graph. We restricted our experiments to those with 50 or 100 nodes, so these were 60 instances in total. The other subset (ogra) consists of instances whose optimal solutions correspond to a dense packing of the tasks for a pre-specified number of processors, i.e., when scheduled on the respective number of processors, none of them will have any idle cycle. The task graphs have $50, 100, 150, \ldots, 500$ nodes, for each size there are 70 instances. In all of those with the same number of nodes, all tasks have the same processing time, and hence only the edge density and communication delays differ. Further, the edge sets of instances with higher density have those of instances with smaller densities as their subsets. Also here we restricted our experiments to instances with 50 or 100 nodes.

We remark here that a common drawback of all, but in particular of the DC and ogra benchmark sets, is, that in most of the instances there exist several arcs with redundant communication delays associated to them. If $(i, j) \in A$ but there is also a path from i to j such that the accumulated lengths of the tasks succeeding i and preceding j on this path is larger than the weight c_{ij} of the arc (i, j), then these communication delays have no effect in practice and the corresponding constraints of the linear programs will not be binding. For the DC instances, all instances with 50 and 100 tasks have arc weights that are redundant in this respect, and the ratio of non-redundant arc-weights varies between only 3 and 29%, and 7 and 50% respectively. In case of the ogra instances, 62 (63) out of 70 instances with 50 (100) tasks have redundant communication delays. In both sets, the percentage of non-redundant ones is however higher on average and lies between 42 and 99%. For the Structure instances, 31 of the 207 instances have redundant arc weights. The ratio of non-redundant ones differs from instance to instance and is spread across the whole interval between 20 and 96%.

While the ogra instance set explicitly specifies how many processors shall be used (up to 16), the parallelism inherent to the DC and also the Structure instances is limited. Hence, we restricted our benchmarks to $P = 2$, 4, and 8 for these sets.

5.3 System Setup, MILP Solver and Time Limits

Our experiments were run on a Debian Linux system with g++ 4.9 and optimization level -O2 on an Intel Core i7-3770T processor running at 2.5 GHz and with 32 GB RAM.

To solve the integer programming models, we used Gurobi[1] in version 6.5.1. In order to reduce side effects as much as possible, we configured Gurobi to use only a single thread and disabled all internal heuristics.

[1] http://www.gurobi.com/

Whenever a MIP is solved using Gurobi, we specified a time limit of 600 seconds. We remark that Gurobi internally uses wall clock (elapsed) time measurement while we display the sustained total CPU and system time (for both, preprocessing and MIP solving) in our figures. Since we used our system exclusively and only ran a single thread, the difference is only marginal. Further, as Gurobi tends to stop some milliseconds before its given time limit and the time required for the preprocessing is negligibly small, the measured CPU and system time never exceeded 603 seconds.

5.4 Preprocessing

All the input tasks graphs are passed to a simple preprocessing routine, in order to derive lower and upper bound positions $\mathrm{lb}(i)$ and $\mathrm{ub}(i)$ for each of the tasks. To this end, the graphs are also appended by an artificial super source and an artificial super sink. Using this approach, no makespan variable is necessary as the starting time variable of the super sink can be used instead.

The preprocessing phase works as follows: At first, a global upper bound $\mathrm{UB_P}$ is established by executing a heuristic that basically resembles the highest-level first with estimated times (HLFET) strategy as presented by Adam et al. [1] but with an adapted processor mapping step to integrate the presence of communication delays similarly as in [44]. More precisely, the tasks are scheduled in non-increasing order of their critical path length (priority). Each task is scheduled on the processor that allows it to start earliest (which depends on the processor assignments of its predecessor tasks and on when the respective processor finishes all other tasks yet assigned to it). Ties are broken such that the task with the larger processing time or the processor with the smaller index is selected. This procedure is once repeated with the DAG that results when reversing all the arcs of the task graph and the better of the two schedules is selected. Secondly, lower bounds on task starting times are derived by transitively propagating the processing times using the precedence relationships in a topologically ordered fashion. This leads to a global lower bound $\mathrm{LB_P}$ (the lower bound on the starting time of the super sink) and minimum distances between dependent task pairs. The latter are then combined with $\mathrm{UB_P}$ in order to transitively propagate the upper bound positions backward starting from the super sink.

The information gained in the preprocessing phase is exploited for all of the models in the same way by:

- Setting the lower and upper bound positions of each task $i \in V$ as the lower and upper bound of the respective starting time variable t_i.
- Fixing any σ-variables for known (i.e. also transitive) precedences.
- Adding communication delay constraints only for arcs $(i, j) \in A$ where $c_{ij} > 0$ and additional simple precedence constraints for the super source and super sink.
- Computing pair-dependent values M_{ij} as described in Sect. 3.3.

5.5 Results

The presentation adheres to the following rules: For each instance set (and size-subset), first the running times of all the models are displayed for either all instances, or (if these are too many) for those that could be solved by at least one of the models. In addition, for all instances remaining unsolved by all models, a plot that displays the lower bounds and best feasible solutions (upper bounds) obtained is added. More precisely, there will be two points for each model and instance, the upper one reflecting the upper bound and the lower one the lower bound computed. This shall give a hint which of the models was closest to proving optimality of a known solution when the time limit occurred. In these figures, the computed lower and upper bounds are normalized to the initially known lower bound after preprocessing $\mathrm{LB_P}$ (giving the base 100) as this value is equal for all models. As a remark, this bound could typically not be improved at the root of the branch-and-bound tree by any of the models, although, with slightly more effort, better lower bounds could be derived from the preprocessing step. Further, the effective size of the models and also their performance is significantly influenced by the derived global lower and upper bound. Hence, in all the bound plots, the upper bound $\mathrm{UB_P}$ is shown (also normalized to the initial lower bound) as 'UB'. In some plots, it becomes apparent that the heuristic schedule is sometimes far away from the optimum solution. On the other hand, there were a few instances solved to proven optimality already by the preprocessing. These instances were filtered out from the displayed figures as no integer program had to be solved.

5.5.1 Graph Structures Instance Set

To display the running times for the 207 Structure instances aesthetically pleasant, we skipped all instances where all models timed out, and split the remaining ones into three parts. Nevertheless, the obtained bounds for the unsolved instances are displayed separately.

For $P = 2$, there were 102 instances that could be solved by at least one model and these are shown in Fig. 1. The picture concerning the performance of the models is however similar in each of the plots: When an instance can be solved, then the x-variables based models (*Core-X*, *Core-X-VS*) are typically the fastest. On some instances, *Core-P* is faster but usually it is rather in the midfield. Except for some single instances, the packing formulation is slower than the three *Core* models and with *2M* several timeout occurred where the other models could solve the respective instance. Six instances were solved by the preprocessing. Among the remaining 99 instances that timed out for all models were nearly all instances with 30 nodes and also many with 21 nodes. For these instances, normalized lower and upper bounds on the objective function value when the timeout occurred are shown in Fig. 2. One can see there that for most of the instances, all models failed to improve on the initial lower bound. On the other hand, the initial upper bounds given could typically be improved by all of the models, i.e., better feasible solutions were found. The random instances could not be solved although the initial lower and upper bounds were relatively close to each other.

For $P = 4$, the number of instances that could be solved by at least one of the models increased to 122. The results, shown Fig. 3, are similar as with

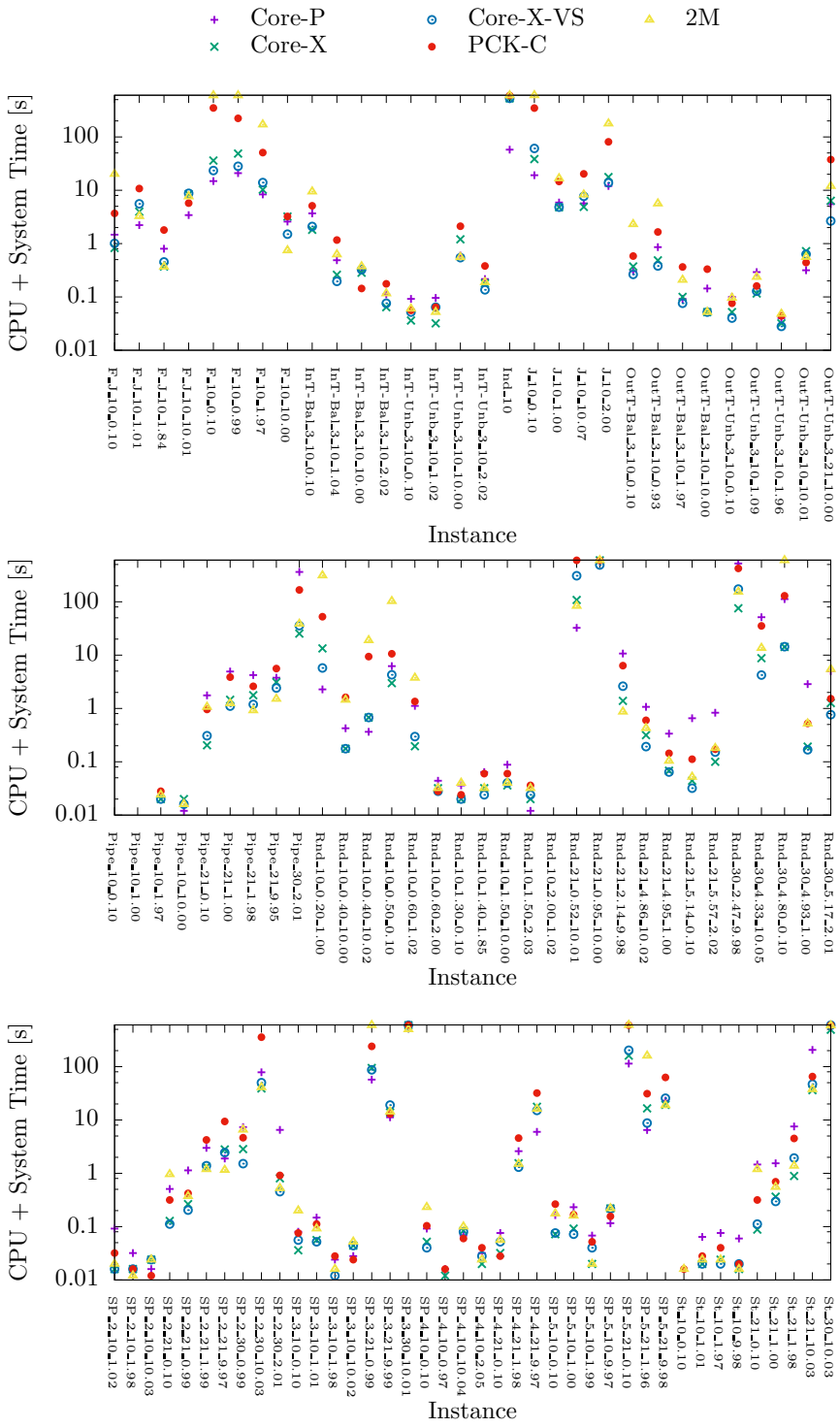

Figure 1: Solution times for the Structure instances and $P = 2$.

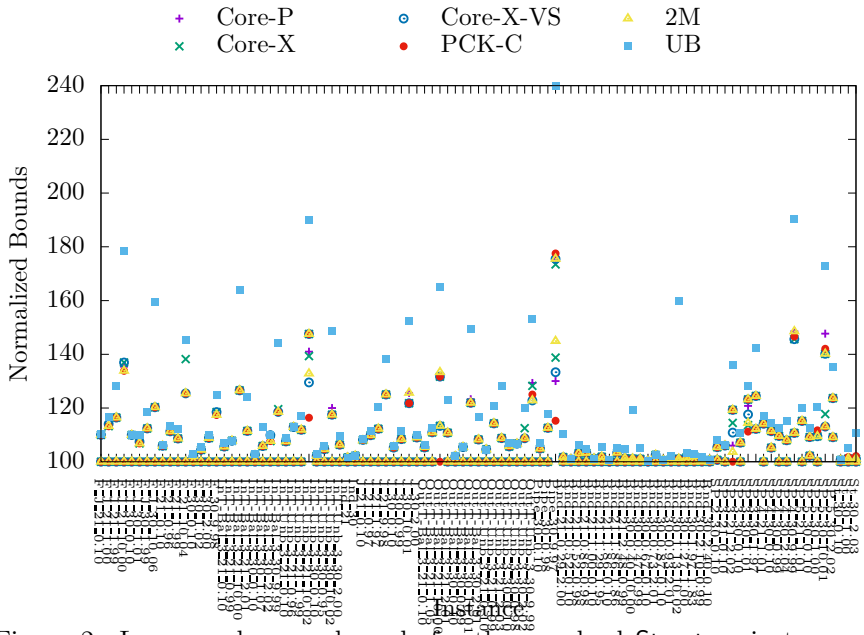

Figure 2: Lower and upper bounds for the unsolved Structure instances and $P = 2$.

$P = 2$, however *Core-P* and also *2M* perform a bit better on average. Further 22 instances could be solved by the preprocessing. For the 63 instances were all models timed out, the final lower and upper bounds on the objective function value are shown in Fig. 4. Again, the best feasible solutions found are typically much better than the initial upper bounds, but no clear trend can be observed.

When further increasing the number of processors to eight, 137 instance can be solved by at least one model and these are displayed in Fig. 5. Again the performance of *Core-P* is improved compared to the $P = 4$-case. On most of the solved instances, it is now the fastest model followed by the x-variable based models and also *2M* which is however not so robust and has several timeouts where the other formulations do not. The majority of outstanding timeouts is however discovered for the packing formulation. Since 36 instances could be solved by the preprocessing, only 34 instances remain that timed out for all the models. For these, the final lower and upper bounds on the objective function value are shown in Fig. 6. It becomes apparent that the lower bounds could now be improved in most of the cases and nearly always by all models. Which model delivers the best bounds varies however from instance to instance. The only noticeable trend is that the packing formulation typically reached the weakest bounds.

5.5.2 DC Instance Set

For the DC instances with 50 tasks and $P = 2$, we see in the upper image of Fig. 7 that in almost all of the cases, either all models fail to find a provably optimum solution within the time limit, or all models succeed in that respect. If only a subset of the models succeeds, then it is usually only the packing formulation

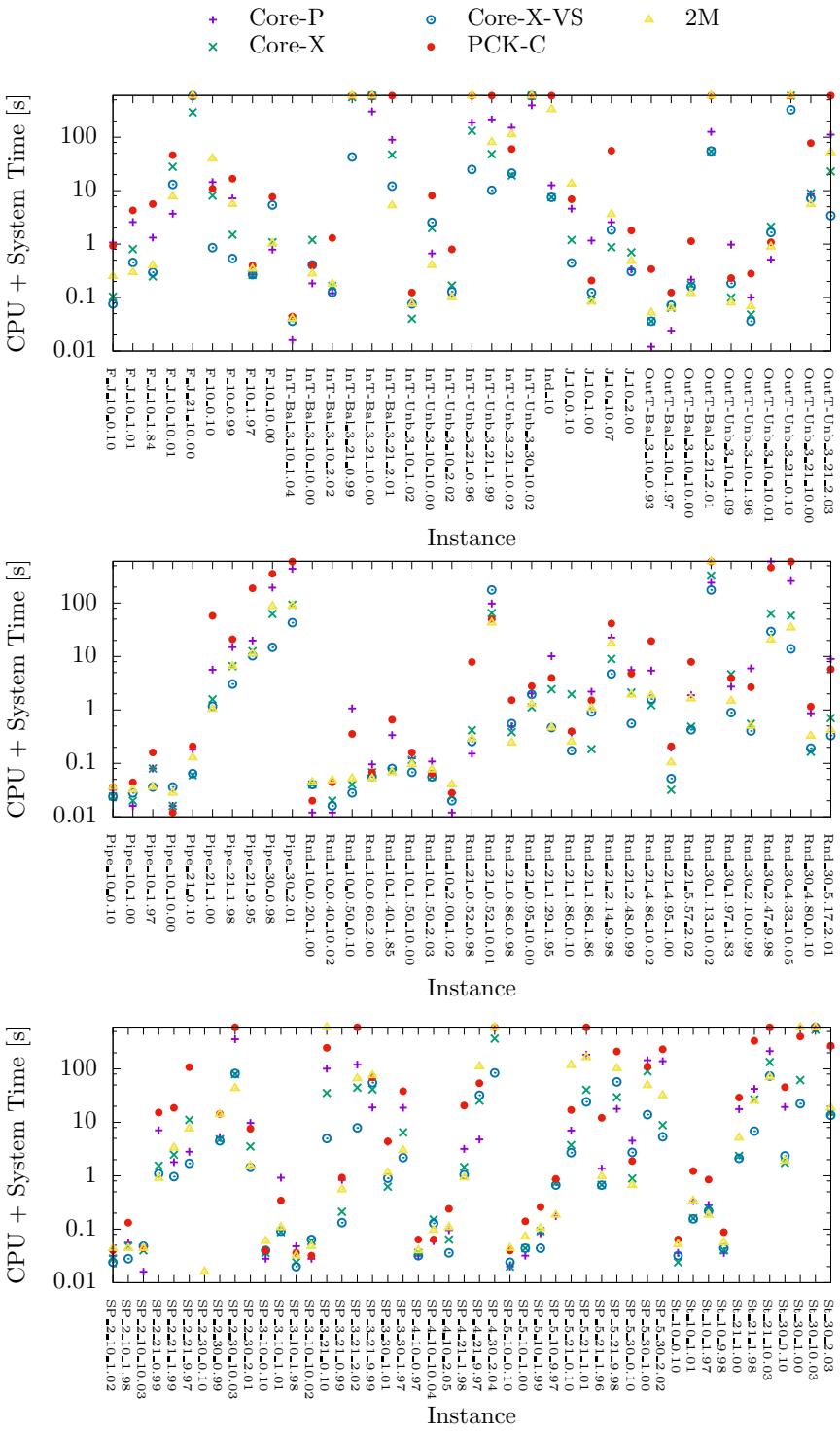

Figure 3: Solution times for the Structure instances and $P = 4$.

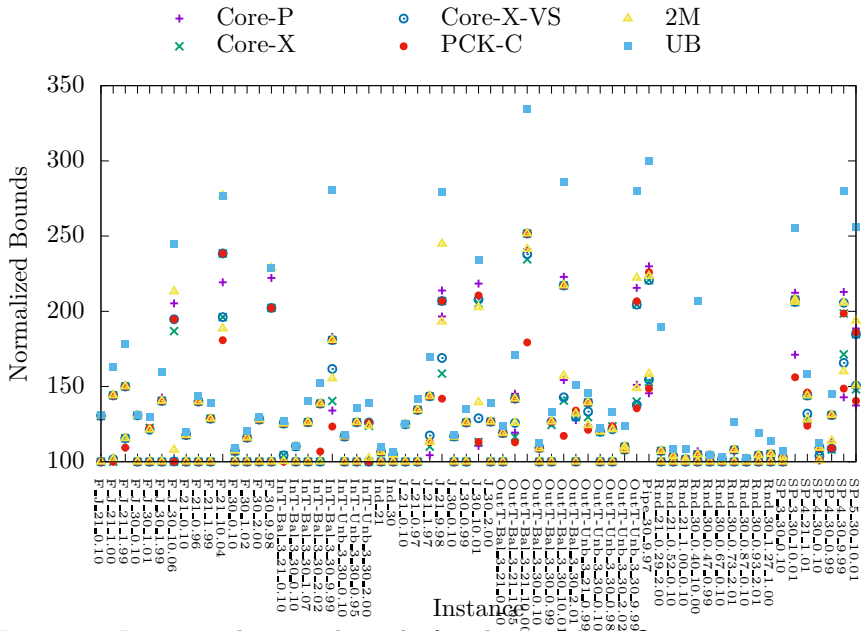

Figure 4: Lower and upper bounds for the unsolved Structure instances and $P = 4$.

that cannot compete with the others, despite the improvements applied here. Also, the packing formulation is usually among the slowest methods in the successful runs, whereas the x-variable-based models (*Core-X*, *Core-X-VS*) are typically the fastest. In most of the successful runs, *2M* and *Core-P* are in the midfield, sometimes faster than the packing formulation, sometimes not and never among the fastest models. The picture is very similar when increasing P to four as is shown in the middle image of Fig. 7. For $P = 8$, the number of instances remaining unsolved by all models increases significantly. Like for the Structure instance set, the relative performance of *Core-P* compared to the x-variable based models is improved (cf. lower image of Fig. 7).

In the upper ($P = 2$) and middle ($P = 4$) images in Fig. 8, one can see that when instances could not be solved by any of the models, then the models *Core-P*, *Core-X*, and *Core-X-VS* usually deliver the best lower bounds, while *2M* and *Packing* are weaker in that respect. Concerning the upper bounds, the x-variable based models perform best. This is also true for $P = 8$ (lower image in Fig. 8), however the better relative performance of *Core-P* in terms of the lower bounds as observed before is confirmed.

For the instances with 100 tasks and $P = 2$, only four out of 30 can be solved by any of the models. The results are given in Fig. 9. We remark here, that even when doubling the time limit to 1.200 seconds, the amount of instances solved increases only slightly, e.g., two more instances can be solved by *Core-X*. Lower and upper bounds for the unsolved instances are shown in the upper image of Fig. 10. For $P = 4$, only once instance can, exceptionally, be solved by *Core-X-VS*, namely t_100_80_3. We refrain from plotting this single instance but again give the lower and upper bounds for the unsolved instances in the middle image of Fig. 10. For $P = 8$, no instance can be solved within the time

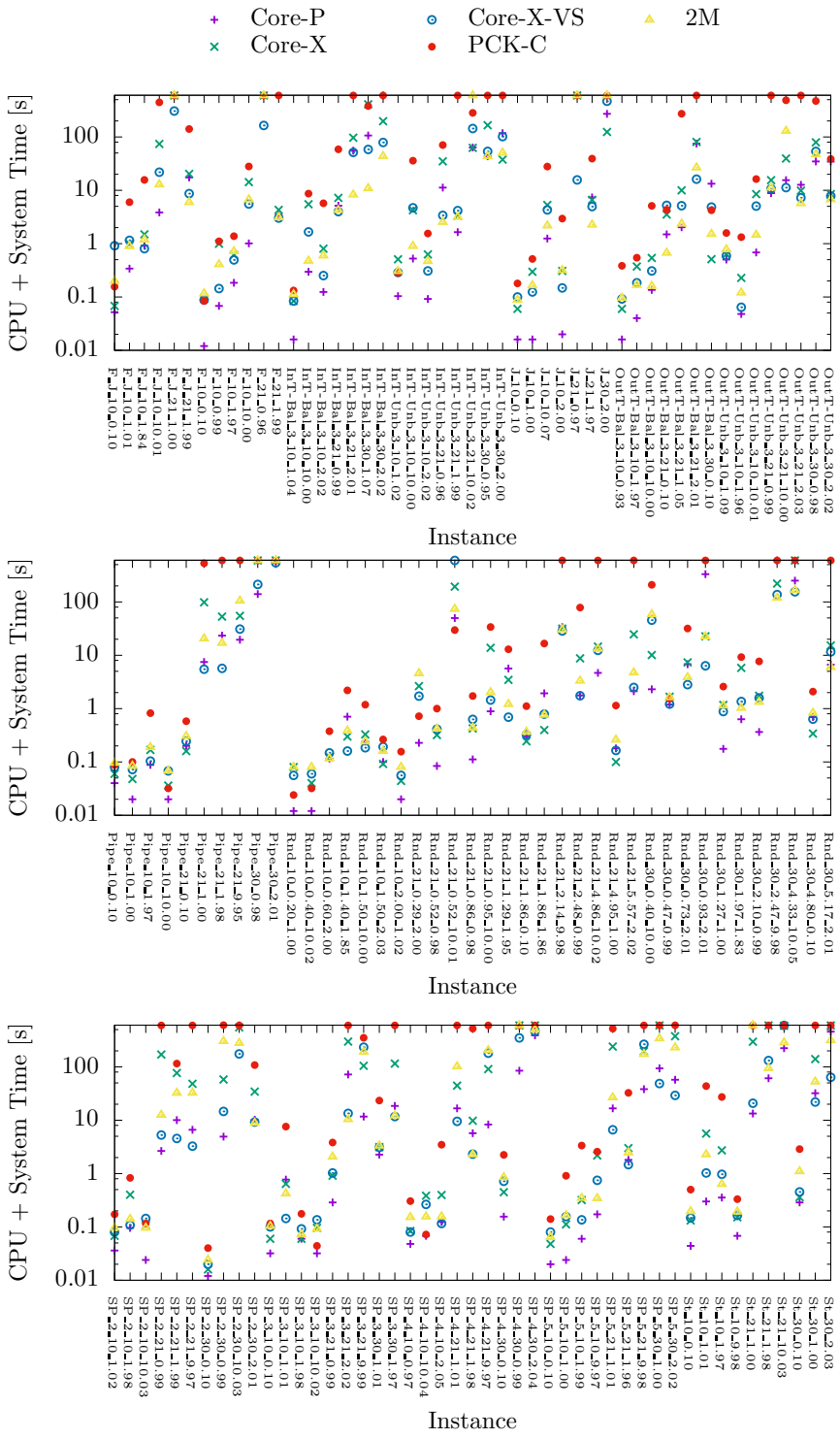

Figure 5: Solution times for the Structure instances and $P = 8$.

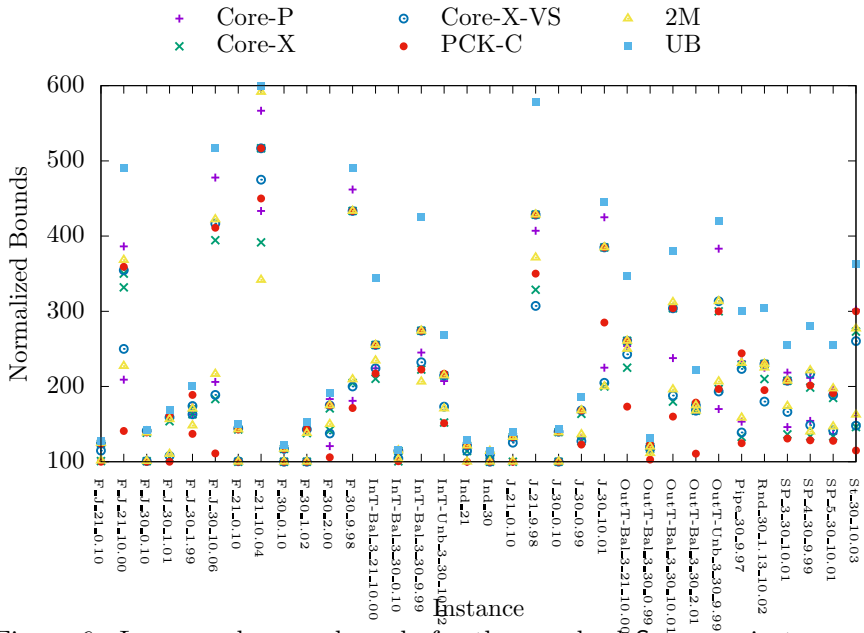

Figure 6: Lower and upper bounds for the unsolved Structure instances and $P = 8$.

limit whence these are also not plotted. Lower and upper bounds are given in the lower image Fig. 10. The relative performance of the models is comparable to the instances with 50 tasks, although the net improvement on the initial lower bounds is significantly worse. For $P = 2$ and $P = 4$, *Core-X* and *Core-X-VS* perform best in terms of the lower bounds, for $P = 8$, however, *Core-P* is superior. Concerning the upper bounds, the x-variable based models typically find the best feasible solutions within the time limit.

5.5.3 ogra Instance Set

For the ogra subset with 50 tasks, two instances were solved to optimality by the preprocessing, 42 instances timed out and only 26 instances could be solved by at least one model. However, most of these 26 instances could not be solved by *2M* or the packing formulation as can be seen in the upper image of Fig. 11. Models *Core-X* and *Core-X-VS* clearly perform best. Due to the special structure of these instances, the initial lower bounds given to the models are typically optimum, and the only task is to *find* the optimum solution. Also here it becomes apparent, that the x-variable based models were typically closest to this goal when a timeout occurred (middle image in Fig. 11). For the instances with 100 tasks, the picture is worse as the models have much more problems in finding good feasible solutions for most of the instances as can be seen in the lower image of Fig. 11. As a consequence, besides one instance solved by the preprocessing, only three instances (not plotted) with $P = 2$ can be solved within the time limit, namely ogra_100_70_2 only by *Core-X*, ogra_100_80_2 by *Core-X*, *Core-X-VS* and *2M*, and ogra_100_90_2 by the same three models.

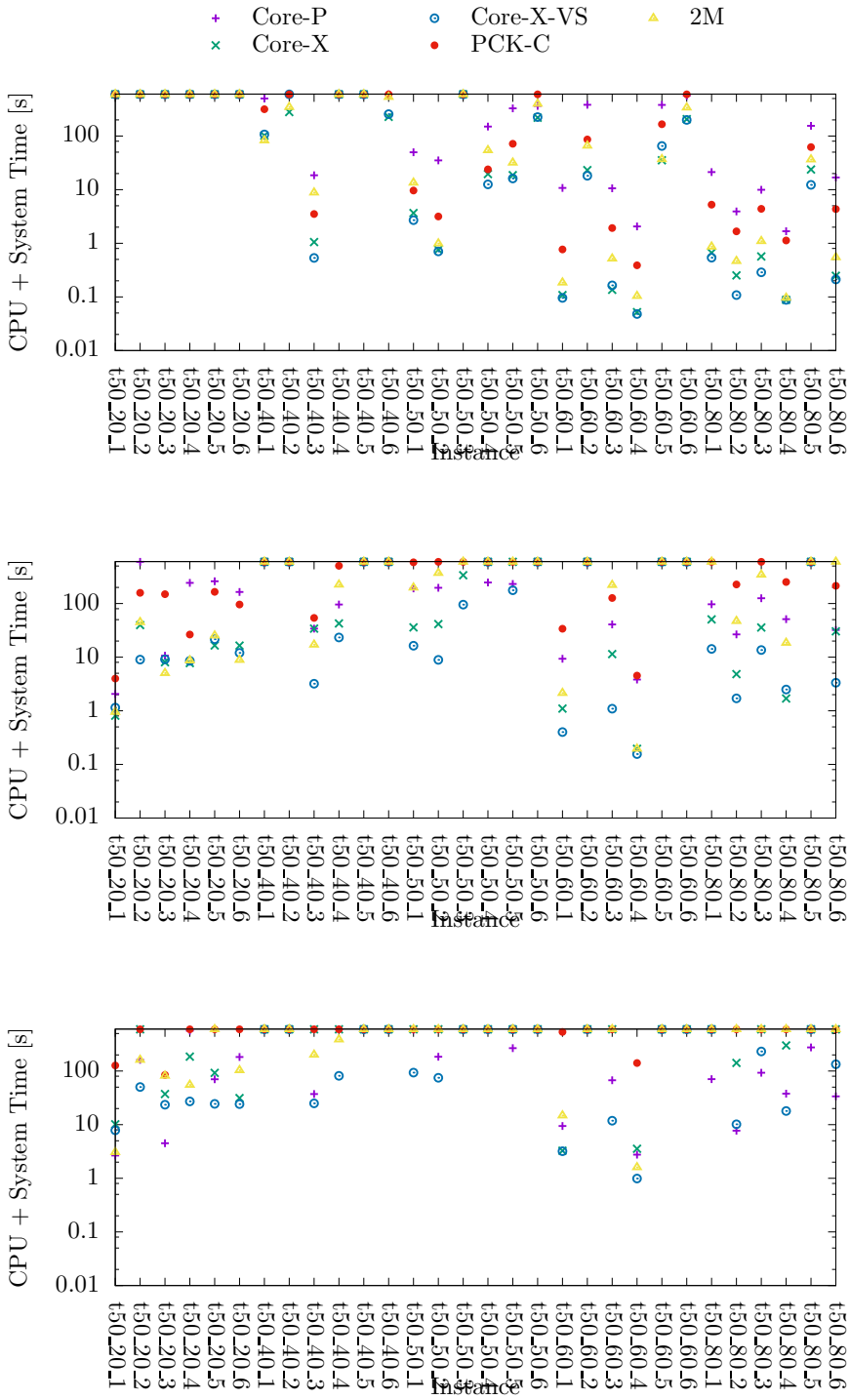

Figure 7: Solution times for the DC instances with 50 tasks and $P = 2$ (upper image), $P = 4$ (middle) and $P = 8$ (lower).

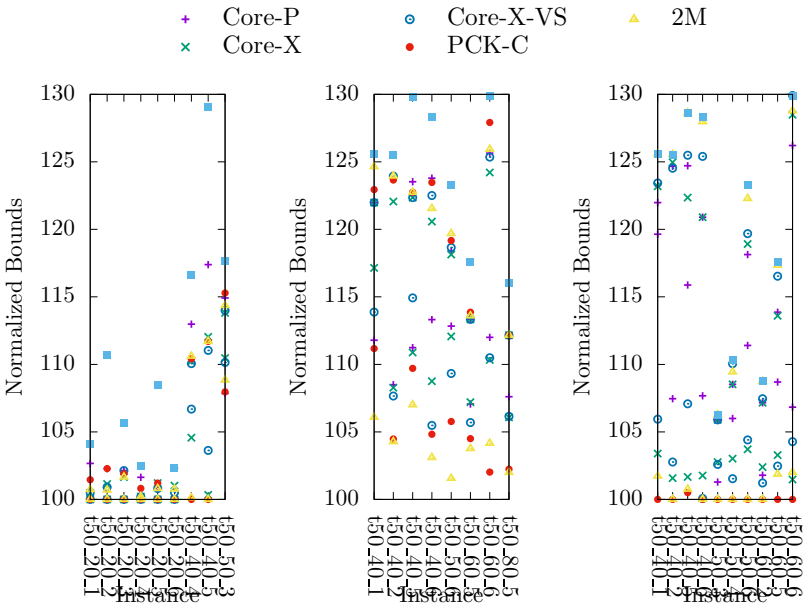

Figure 8: Lower and upper bounds for the unsolved DC instances with 50 tasks and $P = 2$ (upper image), $P = 4$ (middle), and $P = 8$ (lower).

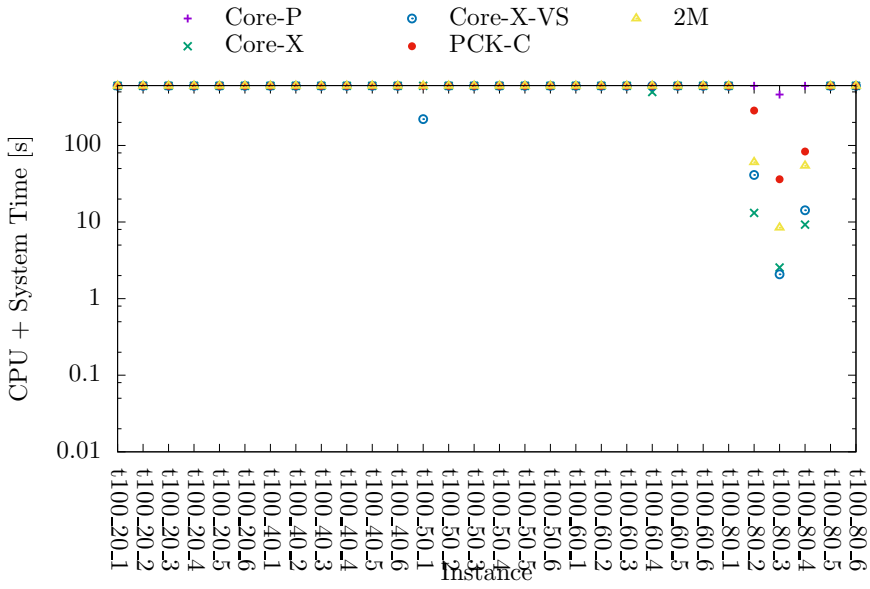

Figure 9: Solution times for the DC instances with 100 tasks and $P = 2$.

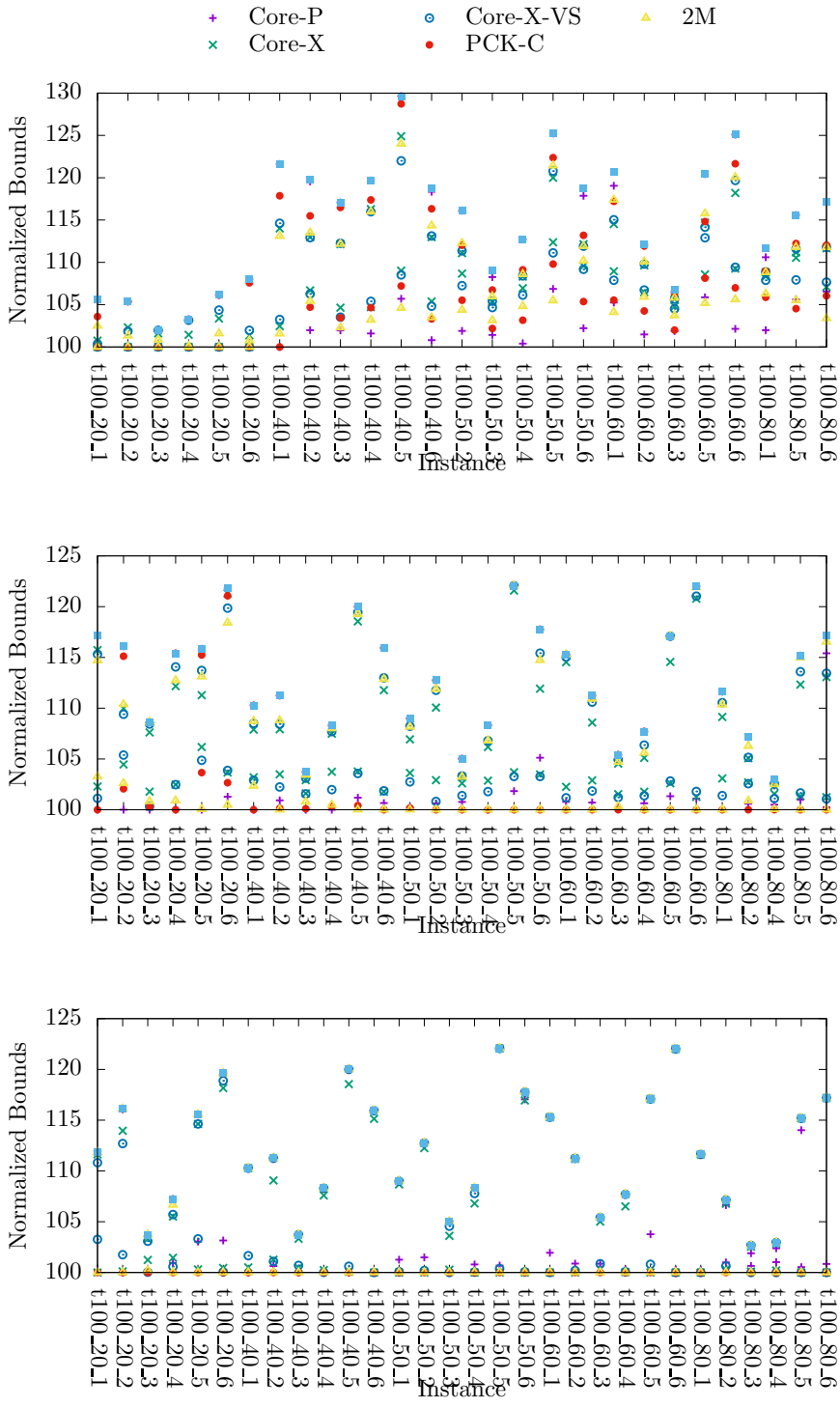

Figure 10: Lower and upper bounds for the unsolved DC instances with 100 tasks and $P = 2$ (upper image), $P = 4$ (middle), and $P = 8$ (lower).

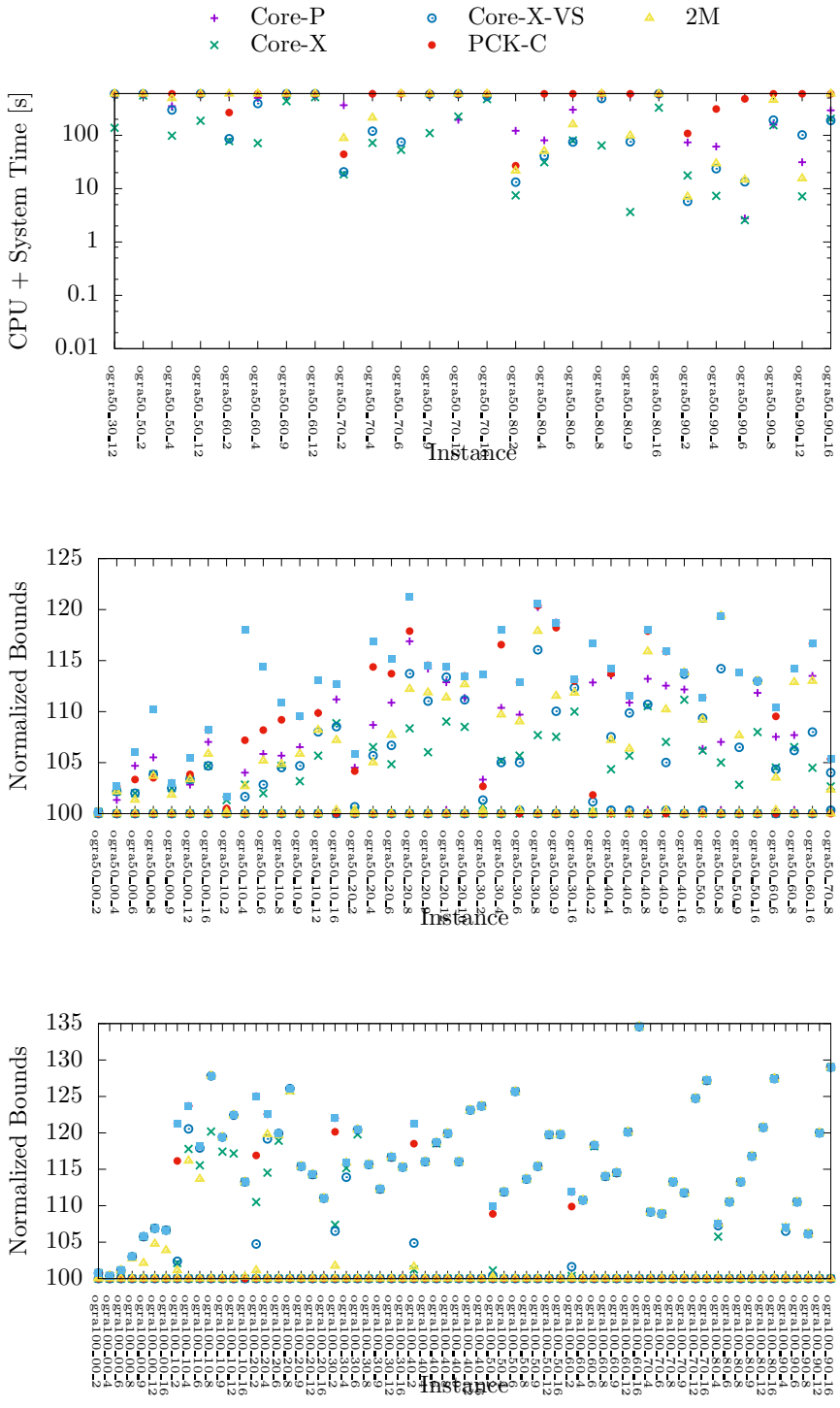

Figure 11: Solution times for the ogra instances with 50 tasks (upper image), and lower and upper bounds for the unsolved ogra instances with 50 (middle) and 100 (lower) tasks.

6 Conclusion and Final Remarks

We have discussed structural improvements of several integer programming formulations for the Multiprocessor Scheduling Problem with Communication Delays. These improvements yield models that are provably smaller while excluding redundant solutions that have been part of the feasible region before. Further, a linearization of the packing formulation has been proposed that introduces a number of additional variables and constraints that is proportional to the number of precedence arcs in the task graph while previous approaches needed a number that is quadratic in the number of tasks.

Despite the avoidance of redundancies especially due to the new equations (26), and the discussed either-or choice between using processor index (p-) and processor index ordering (ϵ-) variables, and processor-assignment (x-) variables, the models presented still have some unwanted symmetries. This is true since, for imposing communication delays, it is not relevant which particular processors two dependent tasks are assigned to, but only whether they are assigned to different processors or not. However, if we refrain from modeling processor assignments explicitly, it becomes very complicated to enforce that all tasks assigned to the same processor are well-ordered while ensuring that no more than the available processors are used.

The experimental results show that the newly introduced models with x-variables (*Core-X* and *Core-X-VS*) are often the most robust choice among the compared models. With an increasing number of processors, the usage of p- and ϵ-variables (as in *Core-P*) appears to be superior. Despite the improvements, it appears that the packing formulation is, in its current form, not competitive in most of the cases. In contrast to that, the model by Ait El Cadi et al. (*2M*) can be competitive but is inferior in terms of the robustness as relatively many timeouts are observed.

References

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Commun. ACM*, 17(12):685–690, Dec. 1974.

[2] A. Ait El Cadi, R. Ben Atitallah, S. Hanafi, N. Mladenović, and A. Artiba. New MIP model for multiprocessor scheduling problem with communication delays. *Optim. Lett.*, pages 1–17, 2014.

[3] J. Baxter and J. Patel. The LAST algorithm – a heuristic-based static task allocation algorithm. In *Proc. Intern. Conf. on Parallel Processing*, pages 217–222, University Park, PA, USA, 1989. Pennsylvania State University.

[4] P. Chrétienne and C. Picouleau. Scheduling with communication delays: A survey. In P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, editors, *Scheduling theory and its applications*, chapter 4. John Wiley & Sons, July 1995.

[5] T. Davidović and T. G. Crainic. Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems. *Computers & OR*, 33:2155–2177, 2006.

[6] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović. Mathematical programming-based approach to scheduling of communicating tasks. Technical Report G-2004-99, Montréal, Canada, 2004.

[7] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, *Proc. 3rd Multidisciplinary Int. Conf. on Scheduling: Theory and Application*, pages 128–135. Ecole Polytechnique, Paris, France, 2007.

[8] T. Davidović, N. Maculan, and N. Mladenović. Mathematical programming formulation for the multiprocessor scheduling problem with communication delays. In *Proc. Simpozijum o operacionim istraživanjima (Symp. on Oper. Res.)*, pages 331–334. 2003.

[9] G. Lj. Djordjević and M. B. Tošić. A heuristic for scheduling task graphs with communication delays onto multiprocessors. *Parallel Comput.*, 22(9):1197–1214, 1996.

[10] H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *J. Parallel Distrib. Comput.*, 9(2):138–153, Jun. 1990.

[11] R. Fortet. Applications de l'algèbre de boole en recherche opérationnelle. *Revue de la Société Française de Recherche Opérationnelle*, 4:17–26, 1960.

[12] S. Fujita and M. Yamashita. Approximation algorithms for multiprocessor scheduling problem. *IEICE Trans. Inform. Syst.*, 83(3):503–509, Mar. 2000.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[14] R. Giroudeau and J. C. König. Scheduling with communication delays. In E. Levner, editor, *Multiprocessor Scheduling, Theory and Applications*, chapter 4. InTech, December 2007.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II, Proc. of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symp.*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[16] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257, 1989.

[17] S. Jin, G. Schiavone, and D. Turgut. A performance study of multiprocessor task scheduling algorithms. *J. Supercomput.*, 43(1):77–97, Jan. 2008.

[18] D. Kim and B.-G. Yi. A two-pass scheduling algorithm for parallel programs. *Parallel Comput.*, 20(6):869–885, Jun. 1994.

[19] S. J. Kim and J. C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. In F. Ris and P. M. Kogge, editors, *Proc. International Conf. on Parallel Processing*, volume 3, pages 1–8. Pennsylvania State University Press, 1989.

[20] B. Kruatrachue and T. G. Lewis. Duplication Scheduling Heuristic (DSH), a new precedence task scheduler for parallel processor systems. Technical Report OR 97331, Oregon State University, 1987.

[21] Y.-K. Kwok and I. Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proc. IEEE Symposium on Parallel and Distributeed Processing*, SPDP '95, pages 36–43, Washington, DC, USA, 1995. IEEE Computer Society.

[22] Y.-K. Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[23] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, Dec. 1999.

[24] Y.-K. Kwok, I. Ahmad, and J. Gu. FAST: a low-complexity algorithm for efficient scheduling of DAGs on parallel processors. In *Proc. International Conf. on Parallel Processing*, volume 2, pages 150–157, Aug 1996.

[25] L. Liberti. Compact linearization for binary quadratic problems. *4OR*, 5(3):231–245, 2007.

[26] C. McCreary and H. Gill. Automatic determination of grain size for efficient parallel processing. *Commun. ACM*, 32(9):1073–1078, Sep. 1989.

[27] N. Mehdiratta and K. Ghose. A bottom-up approach to task scheduling on distributed memory multiprocessors. In *Proc. Intern. Conf. Parallel Processing*, volume 2, pages 151–154, Aug. 1994.

[28] R. H. Möhring, M. W. Schäffter, and A. S. Schulz. Scheduling jobs with communication delays: Using infeasible solutions for approximation (extended abstract). In J. Diaz and M. Serna, editors, *Proc. 4th European Symposium on Algorithms (ESA '96)*, volume 1136 of *LNCS*, pages 76–90, Berlin, 1996. Springer.

[29] V. J. Rayward-Smith. The complexity of preemptive scheduling given interprocessor communication delays. *Inform. Process. Lett.*, 25(2):123–125, 1987.

[30] V. J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Appl. Math.*, 18(1):55–71, 1987.

[31] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors.* MIT Press, Cambridge, MA, USA, 1989.

[32] N. Satish, K. Nadathur, and K. Keutzer. A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors. In *Proc. Conf. on Design, Automation and Test in Europe*, DATE '07, pages 57–62, San Jose, CA, USA, 2007. EDA Consortium.

[33] M. A. Senar, A. Ripoll, A. Cortés, and E. Luque. Clustering and reassignment-based mapping strategy for message-passing architectures. *J. Systems Archit.*, 48(8–10):267–283, 2003.

[34] A. Z. S. Shahul and O. Sinnen. Scheduling task graphs optimally with A*. *J. Supercomput.*, 51(3):310–332, 2010.

[35] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, Feb. 1993.

[36] B. Veltman. *Multiprocessor Scheduling with Communication Delays*. PhD thesis, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands, 1993.

[37] B. Veltman, B. J. Lageweg, and J. K. Lenstra. Multiprocessor scheduling with communication delays. *Parallel Comput.*, 16(2):173–182, 1990.

[38] S. Venugopalan and O. Sinnen. Optimal linear programming solutions for multiprocessor scheduling with communication delays. In Y. Xiang, I. Stojmenovic, Bernady O. Apduhan, G. Wang, K. Nakano, and A. Zomaya, editors, *Proc. 12th International Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP), Part I*, pages 129–138. Springer, Berlin, Heidelberg, 2012.

[39] S. Venugopalan and O. Sinnen. ILP formulations for optimal task scheduling with communication delays on parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):142–151, Jan. 2015.

[40] M.-Y. Wu and D. D. Gajski. A programming aid for hypercube architectures. *J. Supercomput.*, 2(3):349–372, 1988.

[41] T. Yang and A. Gerasoulis. A fast static scheduling algorithm for DAGs on an unbounded number of processors. In *Proc. ACM/IEEE Conf. on Supercomputing*, Supercomputing '91, pages 633–642, New York, NY, USA, 1991. ACM.

[42] T. Yang and A. Gerasoulis. List scheduling with and without communication delays. *Parallel Comput.*, 19(12):1321–1344, 1993.

[43] T. Yang and A. Gerasoulis. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. on Parallel and Distributed Systems*, 5(9):951–967, Sep 1994.

[44] W. H. Yu. *LU decomposition on a multiprocessing system with communication delay*. PhD thesis, U.C. Berkeley, Berkeley, USA, 1984.